

Lecture Notes : 3/20/09

- Srujan.

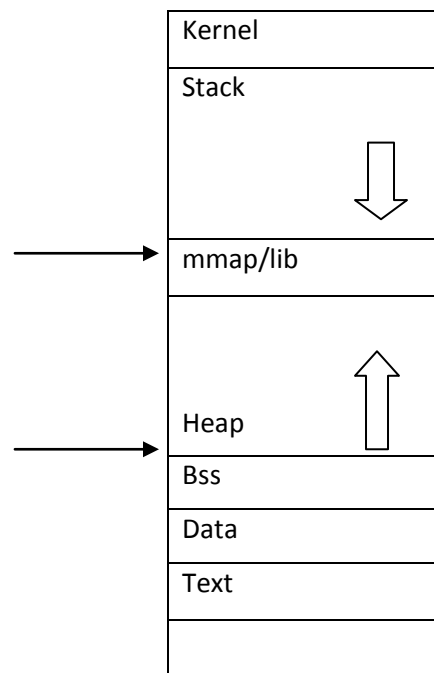
Address space randomization

Buffer Overflow Attack:

Assumptions:

- Attacker knows approximate layout of memory
- Attacker knows exactly the return address of a function in stack
- Attacker knows space between return pointer and buffer

Idea: Take away this knowledge from attacker by Address Space Randomization.



Reduce the probability of attacker to succeed:

- Randomize the start address of stack, mmap/libc, heap.

But Trampoline/Brute force attack on 32-bit architecture would allow attacker to break Address space randomization.

To prevent trampoline attacks shift the text and mmap areas.

Effectiveness of ASR on 32-bit architecture:

Fixed	Bits available for randomization	Page offset bits
4-bits	16-bits	12-bits

4-bits are not randomized so as to prevent fragmentation of virtual address space.

12-bits are page offset bits which cannot be randomized at runtime.

we are left with only 16bits for randomization. Randomizing 16bits wouldn't help much because if attacker gets around 4minutes of attacking time he could break into the code.

Fixes:

1) 64-bit Architecture space

2) **Internal Randomization:**

Stack

- Local variables ordering can be randomized
- Interframe padding (between local variables and return address ?)
- Frame ordering?
- Re-order parameters of a function?
- Intraframe padding

Code Segments

- Function ordering
- Padding
- Instruction choices [randomize the instructions used which provides the same functionality ultimately like use xor in place of sub]
- Polymorphic code?
- Randomize heap
- Instruction set randomization?

Heap

- Interallocation padding ordering

Randomize C library loaded in memory

Above mentioned techniques above for internal randomization might have some performance penalty.

